NORQUEST COLLEGE

CMPT1011

Introduction to Computing

Unit 6: Define Your Own Functions

Learning Outcome

- Explain the use of functions to organize code
- Define your own function
- Demonstrate that variables and parameters are local inside a function

Learning Outcome

- Create functions that return a value or return None
- Create a Docstring for your function

Functions

Functions

- The best way to maintain and develop large program is to construct it from smaller units called **Functions.**
- These functions are easy to modify and reuse it in other programs.
- Function calls are used to execute the block of code.

Benefits

- Functions make your code more readable and easier to maintain.
- It avoids repetition and makes the code reusable.
- It saves time of developers.

Functions

- Just like built-in functions e.g., **min, max, pwr, print, input** etc user can define their own functions to perform a specific task.
- Functions are defined by using a reserved word **def**, followed by a function name and parameters/arguments.
- Return is used to return value from functions.

Python function declaration

- The syntax to declare a function is:

```
def function_name(arguments):
    # function body

    return
```

Python function declaration

- **def** is used to declare a new function.
- **function_name** is used to give any user defined name to a function.
- **arguments** is any value passed to function.
- **return** (optional) value returned from a function

Functions

- The expression in parentheses is called the argument of the function
- A function takes an argument and “returns” a result

Functions

- The result is also called the return value.

Function

```
In [1]: def square(number):
   ...:     """Calculate the square of number."""
   ...:     return number ** 2
   ...:

In [2]: square(7)
Out[2]: 49

In [3]: square(2.5)
Out[3]: 6.25
```

Functions

- Multiple calls can be made to the same function with different values.

Docstrings

Python Docstring

- According to the Style Guide for Python Code, the first line in a function block should be a docstring, a brief description about the function purpose:
- “””Calculate the square of number”””
- These descriptions are strings. Docstrings can be single line or multiline.

Docstrings

We can access these docstrings using the __doc__ attribute.e.g.,

Function_name.__doc__

Example

```
In [1]: def square(number):
   ...:     """Calculate the square of number."""
   ...:     return number ** 2
   ...:

In [2]: square(7)
Out[2]: 49

In [3]: square(2.5)
Out[3]: 6.25
```

print(square.__doc__)

Output: Calculate the square of a number)

Example

```
def sum(a, b):
    """Add two numbers and return the result."""
    return a + b
```

Example

```
def cat(argument1):
    """Summary or Description of the Function

    Parameters:
    argument1 (int): Description of arg1

    Returns:
    int:Returning value

    """

    return argument1

cat.__doc__
```

Hands-on practice

Examples

- Write a program to illustrate a user-define function that calculate the sum of two variables x and y. Also define a docstring in your program.

Examples

```
1  # Program to illustrate
2  # the use of user-defined functions
3
4  def add_numbers(x,y):
5      """ This function calculates the sum of two numbers"""
6      sum = x + y
7      return sum
8
9  num1 = 5
10 num2 = 6
11
12 print("The sum is", add_numbers(num1, num2))
13 print(add_numbers.__doc__)
14
```

Examples

- Write a program to define a user-define function that calculate if the given number is even or odd. Also define a docstring in your program.

Examples

```
1  # Program to illustrate
2  # the use of user-defined functions
3
4  def evenOdd( x ):
5      '''This function calculates if the given number is even or odd'''
6      if (x % 2 == 0):
7          print("even")
8      else:
9          print("odd")
10
11 # Driver code
12 evenOdd(2)
13 evenOdd(3)
14 print(evenOdd.__doc__)
```

Local and Global Variables

Local Variables

- Local variables are variables that can be accessed only inside a function.
- These variables cannot be accessed outside the function.

```
def f():

    # local variable
    s = "I love Intro to Computing in Python"
    print(s)


# Driver code
f()
```

Local Variables

- Now lets see if we can access the variable s outside the function.

```
def f():

    # local variable
    s = "I love Geeksforgeeks"
    print("Inside Function:", s)

# Driver code
f()
print(s)
```

```
Traceback (most recent call last):
  File "./prog.py", line 9, in <module>
NameError: name 's' is not defined
```

Global Variables

- Global variables are variables that are not declared inside a function.
- These variables can be accessed by all the functions.

```
# This function uses global variable s
def f():
    print("Inside Function", s)

# Global scope
s = "I love Intro to Computing in Python"
f()
print("Outside Function", s)
```

Why we use local/global variables?

- What if a Python variable with the same name is initialized inside a function and outside a function.

1. **Will the variable inside a function have some effect on global variable?**

2. **What if we change the value of variable inside a function, will it change the value of global variable?**

Why we use local/global variables?

```
# This function has a variable with
# name same as s.
def f():
    s = "I am local to the function"
    print(s)

# Global scope
s = "I love MLAD program"
f()
print(s)
```

```
I am local to the function
I love MLAD program
```

Why we use local/global variables?

- If a variable with the same name is defined inside a function, it will print the value inside a function and not the global variable value.

- ***Now, if we try to change the value of global variable inside a function, what happens?***

Why we use local/global variables?

```
# This function uses global variable s
def f():
    s += 'GFG'
    print("Inside Function", s)


# Global scope
s = "Outside a function"
f()
```

```
Traceback (most recent call last):
  File "./prog.py", line 9, in <module>
  File "./prog.py", line 3, in f
UnboundLocalError: local variable 's' referenced before assignment
```

The Global keyword

- To make the code work, we need to add “global” keyword the variable.
- Global keyword is only used if we want to do assignments or change global variable.
- Global keyword is not needed for printing or accessing.

The Global keyword

```
# This function modifies the global variable 's'
def f():
    global s
    s += ' GFG'
    print(s)
    s = "Look for Intro to Computing Python Section"
    print(s)

# Global Scope
s = "Python is great!"
f()
print(s)
```

```
Python is great! GFG
Look for Intro to Computing Python Section
Look for Intro to Computing Python Section
```

Try yourself!

```
a = 1

# Uses global because there is no local 'a'
def f():
    print('Inside f() : ', a)

# Variable 'a' is redefined as a local
def g():
    a = 2
    print('Inside g() : ', a)

# Uses global keyword to modify global 'a'
def h():
    global a
    a = 3
    print('Inside h() : ', a)


# Global scope
print('global : ', a)
f()
print('global : ', a)
g()
print('global : ', a)
h()
print('global : ', a)
```

Continue & Break

Continue Keyword

The continue keyword is used to end the current iteration in a for loop (or a while loop) and continues to the next iteration.

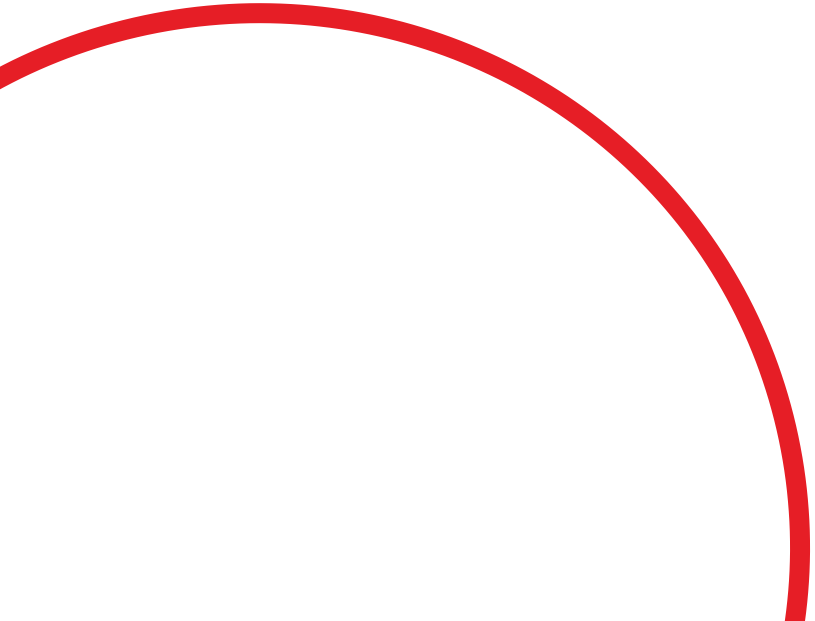
Example

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

```
0
1
2
4
```

Break Keyword

The break keyword is used to break out a for loop, or a while loop.



Example

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

```
0
1
2
```

Hands-on practice

Exercise

1. Write a program to find the first 5 multiple of 6. the format of output should be:

   6 * 1 = 6

   6 * 2 = 12

   6 * 3 = 18

   6 * 4 = 24

   6 * 5 = 30

Hint: Use break statement to break the execution.

Solution

```
# program to find first 5 multiples of 6

i = 1

while i <= 10:
    print('6 * ',(i), '=',6 * i)

    if i >= 5:
        break

    i = i + 1
```

Exercise

2. Write a program to print only ODD numbers from 1 to 10.

Hint: Use continue statement

Solution

```
# program to print odd numbers from 1 to 10

num = 0

while num < 10:
    num += 1

    if (num % 2) == 0:
        continue

    print(num)
```

Questions?

NORQUEST COLLEGE